Dynamic Implicit Surfaces for Fast Proximity Queries in Physically Based Modeling

Bernd Eberhardt\textsuperscript{*}  
Jens Hahn\textsuperscript{†}  
Reinhard Klein\textsuperscript{‡}  
Wolfgang Straßer\textsuperscript{¶}  
Andreas Weber\textsuperscript{¶}  

WSI-GRIS: Arbeitsbereich Graphisch-Interaktive Systeme  
Wilhelm-Schickard-Institut für Informatik  
Auf der Morgenstelle 10  
Universität Tübingen, Tübingen, Germany  
IGD: Fraunhofer-Institut für Graphische Datenverarbeitung  
Rundeturmstr. 6, D-64283 Darmstadt, Germany

Abstract

In this paper we deal with the problem of distance computation between static or dynamic objects, which is an important problem when dealing with the effects of collisions in the context of physically based modeling to obtain a fast numerical integration of the underlying ordinary differential equation. We use an implicit formulation of the (potential) collisions already in the physical laws of the simulated system. The new idea of our approach for distance computation is to use a special implicit representation recently used by Raviv and Elber (in the context of free-form sculpting). In the context of proximity queries this implicit representation has two main advantages: First, any given boundary representation can be approximated quite easily, no high-degree polynomials and complicated approximation algorithms are needed. Second, the evaluation of the corresponding implicit trivariate tensor product B-spline function is very fast and independent of the size of the object.

In the paper we first describe the approach in detail and discuss its space and time complexity. Examples from different areas show the advantages of the approach even for dynamic scenes and objects in practice. Our method is fast enough to maintain a 1000 Hz refresh rate, as is commonly required by haptic rendering, for small objects in very complex environments on standard PC hardware.

Keywords: implicit surfaces, collision detection, physically based modeling, proximity queries, cloth-modeling, haptic rendering

1 Introduction

The problem of collision detection and other proximity queries has attracted substantial research during the last years; see e.g. [16] for a survey. For convex polyhedra algorithms that work in expected linear time have been developed [11, 20, 7]. Other methods that overcome the restriction of convexity have been developed on the basis of various hierarchical bounding volume approaches: oriented bounding boxes used in OBBtrees [10], swept sphere volumes [14], discretely oriented polytopes (k-DOPs) [13]. In [2] also lower bounds for the complexity of a bounding volume approach are proved.

Using these approaches interactive response times are possible on present hardware for models consisting of several thousand triangles. To achieve interactive response times in simulations that are governed by ODEs more sophisticated techniques are needed that allow for fast distance computations in even shorter times.

The benefit of implicit surfaces for collision detection, distance computation and physically based modeling of deformable objects has been known for a long time [24, 21]. In particular an easy inside-outside test involving just a few computations makes the use of implicit surfaces desirable—provided the object of interest can be represented by an implicit function of the desired class. For physically based modeling, the collision model, in its implicit representation, can be incorporated into the formulation of the differential equation (potential field around the object, force field). Boundary conditions of the system (which are given by the collision object) can again be implicitly formulated in the ODE. The numerical integration of the ODE is then fast with large time-steps. The problem however is to find a cheap evaluation of a potential function which represents the collision model as an iso-surface. With the approach described here we can find such a function for arbitrary collision models given as a boundary representation, even for dynamic models.

The use of implicit surfaces for animation purposes involves usually the computation of an animated skeleton. In several settings simulating just the very limited number of particles of the skeleton reduces computation time compared to the calculation of the many vertices of an explicitly given surface. However, examples of such animations are usually “bloppy” and of an artificial nature.

\textsuperscript{*}beber@gris.uni-tuebingen.de  
\textsuperscript{†}jhahn@gris.uni-tuebingen.de  
\textsuperscript{‡}rklein@igd.fhg.de  
\textsuperscript{¶}strasser@gris.uni-tuebingen.de  
\textsuperscript{¶}aweber@igd.fhg.de
For "natural examples" the underlying skeleton is complex again and the advantages of implicit functions are lost. The evaluation of functions becomes quite time consuming as it is not independent of the size of the skeleton. For other standard approaches like algebraic surfaces ([4, Chap. 2]) it is also a problem to come up with good approximations for complex models, and their evaluation is also not independent of the size of the model and thus becomes very costly.

Thus the problem with this setup is to find "THE implicit function" describing the surface of an arbitrarily shaped model. This problem has a certain similarity to the one of finding "the" plenoptic function. The contrast to the plenoptic function is that our object of desire, i.e. the object to be described by a potential function, does not occupy the entire space of its bounding box volume. Other problems arise if the object or some parts of it move in time. This may be a moving arm or leg of a virtual actor etc. The object has to be reconstructed statically (for each single frame) but has also to be reconstructed in motion.

We overcome all these difficulties basically with the single trick described in detail in Sec. 2.1. It is an adaptation of work by Raviv and Elber [22], which they have performed in the context of free-form sculpting, to the problem of collision detection and distance computation between static and dynamic objects. To do this we use a special representation of the given collision model that is quite similar to the one of [22] and that we have modified to assimilate dynamic models. Any given boundary representation of the collision model can be approximated quite easily (see Fig. 2 for a complex example, notice the wrinkles of the trousers) and high-degree polynomials are needed (we use linear B-splines, see also Fig. 2). Thus the evaluation of the distance function as a trivariate tensor product B-spline function is very fast and independent of the size of the collision object. Moreover, in dynamic scenes this evaluation is the same as in the static case with our approach.

2 Implicit B-Spline Tensor Product Surfaces

2.1 Implicit Surfaces in Three Dimensions

The first step in our setting is to reconstruct any given collision model $M$ from a boundary representation as an implicit surface. To do this recall that an implicit surface is given as an iso-surface, i.e. it is the set of all points $(x, y, z)$ with $F(x, y, z) = \omega_0$ for a given iso-value $\omega_0$ and a potential function $F(x, y, z)$. As in [22, 4] we use a trivariate tensor product B-spline function (all with the same degree $d$

$$F(x, y, z) = \sum_{i=0}^{m} \sum_{j=0}^{n} \sum_{k=0}^{n} b_{ijk} B_i(x) B_j(y) B_k(z).$$

One may use different degrees in $x, y$ and $z$, for our purposes taking $d = 1, 2$ or $3$ is sufficient and makes the evaluation fast. Assume the given model $M$ is contained in some bounded volume $V$. Then the trivariate B-spline is defined on a parameter domain $\{x_0, x_1, \ldots, x_m\} \times \{y_0, y_1, \ldots, y_n\} \times \{z_0, z_1, \ldots, z_n\}$, with de Boor-points defined on the grid-points

$$\{x_0, x_1, \ldots, x_m\} \times \{y_0, y_1, \ldots, y_n\} \times \{z_0, z_1, \ldots, z_n\},$$

for real values of $x_i, y_j, z_k$. The main idea in the context of collision detection is now to use a uniform subdivided parameter domain, i.e.

$$x_{i+1} - x_i = \Delta_x, \quad 0 \leq i \leq l,$$

$$y_{j+1} - y_j = \Delta_y, \quad 0 \leq j \leq m,$$

$$z_{k+1} - z_k = \Delta_z, \quad 0 \leq k \leq n.$$

This uniform grid is crucial for the fast evaluation of the distance field.

To define the trivariate B-spline the scalar coefficients $(b_{ijk})_{i=0 \ldots l, j=0 \ldots m, k=0 \ldots n}$ must be determined such that the set $S = \{(x, y, z) : F(x, y, z) = \omega_0\}$ represents the given model $M$.

For the purpose of collision detection it is sufficient to approximate the given model in such a way that guarantees that the original model is inside an $\varepsilon$-neighborhood of the implicit surface. If we use linear B-spline functions $B_i, B_j, B_k$ and have grid-sizes $\Delta_x, \Delta_y, \Delta_z < \varepsilon$, then taking the value $b_{ijk}$ to be the minimum distance from the grid point $(x_i, y_j, z_k) \in V$ to the surface of the boundary representation of $M$ has this property (see Fig. 1).

![Figure 1](image_url)

Figure 1: Representation of an object by a linear implicit B-Spline Tensor Product Surface.

For use in collision detection these linear B-spline functions are sufficient to reconstruct even little details and ensure an easy and fast evaluation of the function and its gradient (i.e. the normal to the surface $S$). To illustrate this have a look at Fig. 2, where we have used $V = [-20, 20] \times [-20, 20] \times [0, 100]$ (inch) and $l = m = n = 200$ and $320$ respectively for the grid resolution of the two reconstructions on the right.

Using higher order B-spline functions we get a smoother reconstructed surface $S$ and we still have a correct inside-outside test by the sign of $F$. The value of $F$ however is not the physical distance in general.

We observe clearly that the grid values $b_{ijk}$ needed to reconstruct the collision model are sparse, since we need just a few of them in a neighborhood of the surface of model $M$. Hence in many cases the supporting grid can be sparse and it is natural to use sparse structures like hash-tables (see Sec. 2.3) to store the grid $G$.

2.2 Evaluation of the Implicit Functions

In order to determine whether a point $P = (x, y, z)$ is inside or outside of the implicit surface $S$ of the object or to get the distance of the point to the object one simply has to evaluate the defining function $F$ at point $P$. For this purpose one has to get the de Boor values at the grid points of the support. The necessary indices can be computed in constant time with a few arithmetic operations. As the support is very small — 8 points for linear functions and 64 for
ues are stored in hash-tables, a great amount of storage cost can be
be used for the implicit representation (see Fig 3). Theoretically
v alues depends on the employed grid-size. Since the de Boor v al-
fraction to the surf ace, also requires more space but does not ha v e
influence on the computation time. Also upper bounds on the approximation error to the
given model \( M \) can be calculated quite easily from the grid size. For storage reasons (and to speed up the pre-computation step) we
usually define the grid only in a neighborhood of the surf ace, as was discussed above. Extending the neighborhood, on which we get the
distance to the surf ace, also requires more space but does not have influence on the computation time.

2.4 Using Levels of Detail

The approximation of objects by an implicit B-spline Tensor Prod-
cube — this operation is also extremely fast. The computation
of the function value requires only a few arithmetic operations.

2.3 Reducing the Necessary Storage Via Hash-
tables

In general for physically based modeling the distance to a surface is
is only of interest in a certain neighborhood of the surf ace. The maximum distance from the surface that is of interest depends on
the velocity of the simulated objects and the time steps of the sim-
ulations. If this maximum distance is \( m \) we thus only have to store
\( O(ms) \) points in our grid for a surface of size \( s \). Using hash ta-
bles we can actually store this subset of the necessary points with
storage costs \( O(ms) \) without losing the property of fast evaluation
of our defining function: access to hash tables does not only have
(expected) asymptotic time complexity \( O(1) \) but is also very fast in
practice.

Remark. Strictly speaking we get an approximation of the dis-
tance in a certain neighborhood of the implicit surf ace. Getting bet-
ter approximations can be achieved by a finer grid, which requires
more storage space but does not have any influence on the computa-
tion time. Also upper bounds on the approximation error to the
given model \( M \) can be calculated quite easily from the grid size. For storage reasons (and to speed up the pre-computation step) we
usually define the grid only in a neighborhood of the surf ace, as was discussed above. Extending the neighborhood, on which we get the
distance to the surf ace, also requires more space but does not have influence on the computation time.

2.5 Moving the Environment or Dynamical Collisi-

2.5.1 Implicit surfaces in four dimensions

If our collision object \( M \) is moving, we have its boundary repre-
sentation given at certain key-frames. For each key-frame we may
reconstruct the implicit surface \( S \) of our model. This gives a has-
table grid at each defined time-step of the key-frames. Having a pre-computed animation sequence to be used as a dynamic collisi-
ion model we can reuse the entire setting if we regard the 3D-space
and one time dimension as 4-dimensional time-space. Thus we can
take 4-variate tensor product B-splines. Having linear B-splines we
are similar to linear key-frame animation of the environment, using
cubic B-splines our key-frame animation is smooth. The unified 4-
dimensional time-space could also be used to represent morphings
between two objects instead of the movement of one object.

2.5.2 Implicit surfaces in three dimensions with func-
tional values

Recall that in our setting of a pre-computed distance-field we have to store at each fixed(!) grid-point \( (x_i, y_j, z_k) \) in our bounding box
volume the distance to the collision model. It is now simple to store
a function depending on time instead (and possibly other parame-
ters). This implies that the de Boor points for our trivariate tensor
product B-spline function \( F \) are now time dependent real values \( b_{ijk}(t) \).
The advantage of this approach is that, even for a dynamic collision scene stored in this way, the inquiry for collision is of the same negligible price as for static scenes. Usually one uses linear interpolation between key-frames. These functions are of a simple nature and can be compressed quite easily to store just the necessary values.

Remark. In our current implementation we have a somewhat special treatment of the time dimension in order to compress data: In moving scenes very often only parts of the environment are moving, whereas other parts remain static or have a simpler movement such as a linear one. So we assign to every 3-dimensional grid point a one-dimensional function that allows for the compression of piecewise linear parts.

2.6 Further Possibilities: Massive Models and Parallelization

In the recent survey on collision detection by Lin and Gottschalk [16] the following two problems in connection with collision detection are stated (among others): Collision detection for massive models and possibilities for parallelization.

Existing approaches like the hierarchical bounding boxes are very memory intensive [23]. For large objects consisting of several million triangles the auxiliary structures cannot be held in main memory and relatively complicated secondary memory techniques must be used [23].

Our approach to collision detection and proximity queries has very good properties with respect to both problems:

Massive models: Even if the environment is so large that it does not fit into main memory, building the hash-tables can be done by considering one triangle after the other. Thus only parts of the model have to be present in main memory

By splitting the hash tables into overlapping ones of a size such that any of them fits into main memory gives a quite straightforward implementation of our approach for massive models. This is much easier than the secondary memory techniques used in the IMMPACT system [23] for a traditional bounding box approach.

Parallelization: After the pre-processing step is performed we only have to read from the data structures that were built from the environment. In addition to the quite obvious parallel sub-algorithms this fact should allow a very efficient parallelization by multi-threading on a shared memory machine.

Also the pre-processing step is suitable for parallelization, because there are few write operations involved in it, which require locking of the shared memory. The synchronization requirements are rather low. Thus good speed-ups can be expected by parallelization of our system on shared-memory systems.

2.7 Performance Results

We implemented the system in C++. For the performance critical part of access to large hash tables we currently use the implementations of LEDA [19, 18].

On a 500 MHz Pentium III PC an evaluation of the implicit function took about 24 \( \mu \text{sec} \) if the hash-tables were filled with 88414 entries. With 288660 entries in the hash tables the corresponding evaluation took about 44 \( \mu \text{sec} \). For smaller models in a bounded region (such as the figurine used in Sec. 3.1), for which we did not need the sparse data structure of hash-tables but could use C arrays to represent the grid, a function evaluation took about 2 \( \mu \text{sec} \).

We obtained these measurements by considering the implicit surfaces used in Sec. 3.1 with different grid resolutions and then taking the timing for 100000 implicit function evaluations at random points close to the surface.

Although access to hash-tables does not have uniform time requirements due to possible hashing collisions, it is quite reliable in a certain range. This is an important property of this method and an additional advantage to its general performance, especially when applied to haptic rendering, cf. Sec. 4 and [17, Sec. 2].

3 Applications

3.1 Cloth Modeling

Problems of collision detection for cloth modeling have been our major motivation for this work. The computation times for cloth modeling is vastly influenced by the choice of the integration schemes, appropriate setting of internal damping and other relatively subtle factors. We cannot even sketch these problems here but have to refer to the literature [1, 6, 8].

Until quite recently the computation time for cloth/surface collision detection in cloth modeling has been considered rather negligible compared to solving of the underlying differential equations. So most implementations of cloth/surface collision detection algorithms in cloth modeling (such as in [1] or [8]) use hierarchical bounding box tests. However, also in this setting the use of a smooth force field in contrast to a non-smooth collision detection test is highly desirable, since higher-step sizes for the implicit integration schemes can be achieved. An implicit reconstruction of the obstacles can be used to define a sufficiently smooth force-field; the step-sizes of our integration schemes improved by about a factor of 10 when switching from non-smooth collision response to the smooth one given by a force field on the implicit surfaces.

To illustrate this we computed a simple poncho on a figurine, whose arms are rising, cf. Fig. 4. The integration method used for this simulation was adapted from [8]. The overall simulation time is dominated by the time for numerically solving the system of ordinary differential equations (ODEs). This is still true for the method given in [1], which is suitable for a simplified cloth-like material and faster than the general method we use at the moment. However, recently a very interesting new method for simulating another simplified cloth-like material has been presented in [6]. This method is faster than the given in [1] by a factor of 10, and it seems to be possible to achieve even further speed-ups by additional analysis and tailoring of suitable cloth-like material. For these fast techniques fast proximity queries to the obstacles are even more important. So our method is applicable for this new context and the improved performance will become more and more important in the near future.

3.1.1 Computational results

Since a single distance computation is extremely cheap (cf. Sec. 2.7) and its use is scattered in the integration function, we cannot measure directly the required time for the distance computations. Instead we counted the number of invocations of the implicit function for the simulation process. From this value we can deduce the required overall time for this task taking the values given in Sec. 2.7, measured in a loop at random points close to the surface. As the computational costs of an implicit function can be assumed

\footnote{For this purpose also other distance computations can be used in principle. Notice however, that in the context of cloth modeling non-convex settings have to be considered. Moreover, in general distance computations are about an order of magnitude slower than collision detection tests, when the same underlying bounding-volume technique are used [14].}
to be almost independent of the argument, this method should give quite adequate results.

For the result of the simulation given in Fig. 4 a simulation time of 0.96 sec was necessary. The poncho consists of 2484 particles. For the force field computation 9664878 calls to the implicit function had to be made during the entire integration. Gradient computations are only performed if a particle is in a certain proximity of the surface (about 1/2 inch in our current setting). The number of gradient computations was 8113161, i.e. about 8% of the number of calls to the implicit function. The figurine is small enough so that C arrays can be used to store the grid for the implicit function instead of hash tables (on a PC with 512 MB RAM), so that the access times for C arrays given in Sec. 2.7 have to be multiplied by the number of calls. Thus our estimation for the time needed for the distance computations is 192 sec. The overall computation time was 7200 sec. The very high-number of implicit function invocations is due to the use of the force-field for collision response. Using a non-smooth collision response would decrease the number of calls to a collision detection method, but for the price of much smaller step-sizes (about a factor of 10, i.e. 0.0002 against 0.002 using the force-field in our example). It is therefore important to have a very fast distance computation method that can be called frequently from the ODE-function (formulated as a potential field for the collision model) in order to support a good integration method and obtain large time-steps. Combining our fast distance computation method with the method of [6] seems to be a promising possibility towards interactive virtual dressing.

In the context of virtual fashion the scenario of using a pre-computed movement of a figurine is quite natural. Users might want to visually examine the fit of different types of cloth to a figurine (which might correspond to 3-dimensional scans of themselves). Having a specific movement in the figurine facilitates the comparison between the different clothes.

3.2 The revolving Lottery Machine

This example illustrates the ability to reconstruct detailed scenes containing several objects of greatly varying dimensions and use this reconstruction for collision detection purposes. The picture of the reconstruction given in Fig. 6, was obtained using marching cubes to extract the iso-surface (with $\omega_4 = 0$) of the trivariate tensor product B-spline function. It is only included to visualize the quality of the iso-surface used for collision detection. The fragments in the pictures are due to the coarse resolution in the used marching cubes extraction.

The idea to simulate the moving balls in the mixing lottery machine is now as follows: Use the implicit surface reconstruction of the relatively complicated geometry of the lottery machine to represent one rotation. Use the novel proximity query techniques described in this paper for interaction of the balls with parts of the rotating lottery machine. For the self interaction of the balls their immediate implicit surface representation as spheres can be used.

Notice that by storing one rotation of the lottery machine we can easily simulate the entire movement of it involving several forward and backward movements: we just have to transform the simulation time to the time scale of one rotation accordingly. All that is needed for our approach is that the movement of the lottery machine is not influenced by the collisions with the balls, an assumption that clearly holds.
are widely used in one form or the other and which have (at least) been that only a very restricted (“blobby”) class of objects could be modeled. Using the approach of this paper, arbitrary surface representations of (dynamic) objects can be very well approximated as a potential function. Since the evaluation of the potential function is so cheap it can be integrated in the formulation as boundary conditions (force field or potential) of the differential equation in physically based modeling. This makes this technique attractive to gain large time-steps in the numerical treatment of the ode. We got a speedup in the time-steps for cloth-modeling by a factor of 10!

Using the sparse data type of hash tables the required storage of the implicit representation is only \( O(s) \), where \( s \) is the surface area of the objects. Hence it is independent of the given resolution of the boundary representation.

The short theoretical and practical computation time for collision detection can be attributed to the following facts:

- the finite support of the used implicit surface representations;
- the constant access time to the hash-tables used in the representations.

This constant time access has a clear advantage over the tree approaches in the various “hierarchical bounding volumes”, which are widely used in one form or the other and which have (at least) logarithmic costs [15, 3, 12, 10].

Our approach is not restricted to convex polyhedra as are the linear time methods for convex polyhedra, see e.g. [11, 20].

Having the information of a distance is good for collision response in many settings, not only for physically based modeling. This information could also be used for haptic rendering. For haptic rendering the maintenance of a reliable high refresh rate (1000 Hz) is different from the response to the colliding object. Most of the other approaches are symmetric with respect to the colliding objects. They do not take advantage from the asymmetric situations. Our approach to collision detection should not be seen as a general method. However, since these asymmetric settings are quite common, we have developed a new and faster collision detection and distance computation method for many scenarios.

### 4 Conclusions

The idea of using implicit surfaces for collisions detection is well known, cf. [5, 4]. However, a major problem of this approach has been that only a very restricted (“blobby”) class of objects could be modeled. Using the approach of this paper, arbitrary surface representations of (dynamic) objects can be very well approximated as a potential function. Since the evaluation of the potential function is so cheap it can be integrated in the formulation as boundary conditions (force field or potential) of the differential equation in physically based modeling. This makes this technique attractive to gain large time-steps in the numerical treatment of the ode. We got a speedup in the time-steps for cloth-modeling by a factor of 10!

Using the sparse data type of hash tables the required storage of the implicit representation is only \( O(s) \), where \( s \) is the surface area of the objects. Hence it is independent of the given resolution of the boundary representation.

The short theoretical and practical computation time for collision detection can be attributed to the following facts:

- the finite support of the used implicit surface representations;
- the constant access time to the hash-tables used in the representations.

This constant time access has a clear advantage over the tree approaches in the various “hierarchical bounding volumes”, which are widely used in one form or the other and which have (at least) logarithmic costs [15, 3, 12, 10].

Our approach is not restricted to convex polyhedra as are the linear time methods for convex polyhedra, see e.g. [11, 20].

Having the information of a distance is good for collision response in many settings, not only for physically based modeling. This information could also be used for haptic rendering. For haptic rendering the maintenance of a reliable high refresh rate (1000 Hz) is different from the response to the colliding object. Most of the other approaches are symmetric with respect to the colliding objects. They do not take advantage from the asymmetric situations. Our approach to collision detection should not be seen as a general method. However, since these asymmetric settings are quite common, we have developed a new and faster collision detection and distance computation method for many scenarios.

### References


